A Single-chip Solution for the Secure Remote Configuration of FPGAs using Bitstream Compression

Jo Vliegen*†, Nele Mentens*†, Ingrid Verbauwhede*
*KU Leuven, ESAT/COSIC; Kasteelpark Arenberg 10, Leuven, Belgium; & iMinds
†KU Leuven @ KHLim, ACRO-ES&S; Agoralaan B, Diepenbeek, Belgium
Email: firstname.lastname@{esat.kuleuven.be}

***Abstract*—This paper presents a system that allows the secure remote configuration of an FPGA, which is assumed to be the only device in the secure zone. This means that no security critical information passes over the borders of the FPGA chip, reducing the opportunities for an attacker to break the system. In particular, bitstream compression in combination with partial reconfiguration is used to avoid the use of an external memory for the storage of the bitstream. Additionally there is no need for an external processor for the transfer of the bitstream. Nevertheless, our solution contains a mechanism that verifies the integrity of the complete bitstream before starting the configuration. This prevents attempts to load unqualified bitstreams and reduces the downtime. The integrity check, the decryption, the authentication of the origin and the freshness check of the bitstream are performed inside the FPGA while its current configuration is still active. The contribution of this work is that it presents the first complete working system for the secure remote configuration of FPGAs, consisting of a single FPGA chip and an initiating server, given that the integrity of the complete bitstream is verified before configuration. This paper gives details on the overall system and the FPGA architecture, which have been implemented and tested.**

I. Introduction

Electrical devices can be shipped with an FPGA. This comes at higher cost on material, but the lifespan of such devices can be extended. An FPGA driven electrical device can be reconfigured to solve potential issues in hardware or software. Additionally, these devices can be tailored to the users' needs, by adding and removing features on the FPGA.

Various research projects have studied remote reconfiguration of FPGAs. The focuses of these studies can mainly be divided into two groups: those with a focus on IP cores and those with a focus on general FPGA reconfiguration. Both groups have overlapping requirements such as: communication of new configurations, and securing this communication. Differences in both groups are mainly towards interconnectivity of a reconfigurable partition and the static partition in which the former resides. Within the group focused on IP cores, the interconnectivity is very strict. Also the available reconfigurable resources in a reconfigurable partition are defined very precisely. In the group focused on general FPGA reconfiguration, these restrictions are relatively loose in the sense that the designer has more resources available to implement the design and the interconnectivity is more flexible. This work focuses on the latter group.

With the focus on general FPGA reconfiguration there are **no** functional connections between the static partition and the reconfigurable partition. Also the reconfigurable partition is preferably as large as possible. This unavoidably reflects in large partial bitstreams which results in large storage requirements in the static partition. Every configuration bit, determining the behavior of the reconfigurable partition, has to be rewritten. For Block RAMs (BRAM) also the initialization values of the memory, which tend to have high entropy, are explicitly stored in the bitstream. This implies that a configuration of an FPGA can never be stored on the on-chip memory of the same FPGA.

This work provides a solution for general FPGA reconfigurations of FPGAs in the field which are connected to the Internet. The partial bitstream never leaves the FPGA chip. This reduces the opportunities on which an attacker can attack the scheme. The single-chip solution is implemented and tested, for which the results are also taken up in this work.

In sect. II an overview is given about published work on remote FPGA reconfiguration and bitstream compression. The solution to achieve a single-chip solution is explained in sect. III, while sect. IV explains the prototype implementation. This work ends on sect. V which draws conclusions and discusses future work.

II. Related Work

Within the scope of this work two different fields of research are combined: 1) the (secure) remote reconfiguration of FPGAs, and 2) the compression of partial bitstreams. This section gives an overview of published work on both topics.

A. Remote reconfiguration of an FPGA

In 2004, Hübner et al. describe a real-time, dynamic reconfiguration of an FPGA [1]. Their bitstream is compressed using Lempel-Ziv-based algorithms and then delivered to the FPGA in an off-line setting. While their focus was on real-time behavior and efficient decompression, our solution combines these features with remote configuration and security.

A system for remote self-reconfiguration of an FPGA is presented by Castillo et al. [2], which downloads a bitstream from a TFTP server which is stored in external ZBT SRAM.

978-1-4799-2079-2/13/$31.00

In 2008, Gonzalez et al. published [3] in which they present an IP core based solution to assist the processor in cryptographic calculations. Their system provides a single reconfigurable partition in which an implementation of the requested cryptographic algorithm can be configured. The work does not state whether the partial bitstreams are communicated or stored in an external component.

Devic et al. present in [4] and [5] a secure protocol and an implementation which offer remote bitstream updates. They foresee countermeasures against replay attacks and bitstream spoofing and provide a solution without the need for external memory. The difference with our solution is that we do not perform on-the-fly configuration, but we perform an integrity check on the complete bitstream before configuration. This makes it more challenging to obtain an implementation without external memory but prevents configuration incomplete and/or corrupted bitstreams.

The work of Kepa et al. [6] presents a secure reconfiguration controller. Using SeReCon, they allow updating and interchanging IP cores.

A new protocol for secure, remote updates of FPGA configurations is presented by Drimer in [7], providing confidentiality, integrity and freshness of the bitstream. A non-volatile memory is used for storing the incoming data either encrypted or already decrypted.

In [8], Braeken et al. describe an on-line system handling incoming bitstreams, providing data confidentiality and authentication, and mutual explicit key authentication. Their work uses external flash memory to store the incoming bitstream, which means the FPGA goes off-line between configurations which could threaten the security.

B. Bitstream compression

In [9], Pan et al. present a new technique: Difference Vector compression, which achieves data compression by sending only the differences between a configuration frame and a beneficiary frame. They extend this technique by reading back frames from the current configuration, which are used as a reference to build the Difference Vector.

Koch et al. evaluate three techniques: run-length encoding, Lempel-Ziv algorithms and Huffman compression [10]. They compare these techniques on a set of benchmark bitstreams both on compression ratio and resource cost of the decompressor.

Haiyun and Shurong have implemented in [11] the LZW algorithm [12]. Their work achieves a compression ratio of 43.69% for partial bitstreams.

The work of Ştefan and Coţofană [13] focuses on bitstreams for the Xilinx Virtex 4 devices. They use several techniques and combinations of techniques to achieve bitstream compression; and they discuss the accompanying hardware implementations to perform the decompression. They draw two main conclusions: 1) the internal organization of bitstreams is likely to change between FPGA families and 2) synthesis tools produce bitstreams with the ratio of "0" and "1" symbols as the main source of redundancy. The latter turns the focus on the simpler compression methods.

A combination of run-length encoding and bitmask-based compression is used in the work of Qin et al. [14]. They outperform other compression techniques, while maintaining high speed decompression.

Note that most research on bitstream compression is performed to reduce the configuration time of the FPGA, while our goal is to avoid the need for an external memory.

In summary, many features of our solution have been implemented in different settings and with different constraints, but our work presents the first working system combining secure remote configuration with bitstream compression to obtain a single-chip solution, for the general reconfiguration of an on-line FPGA.

III. Single-chip solution

To achieve a single-chip solution for a general FPGA reconfiguration, this work uses partial reconfiguration and bitstream compression. Additionally, the communication of the FPGA updates is secured against various types of attacks.

A. Partial reconfiguration

Configuring an FPGA generally occurs in one iteration. This implies that altering a configuration results in reconfiguring the complete FPGA, independent of the size of the update. Partial reconfiguration has been introduced to avoid this inefficient method. Using the partial reconfiguration techniques the resources of an FPGA get divided in two or more partitions: a single static partition and one or more reconfigurable partitions. The former stays configured and working while the latter can be altered. The bitstream which reconfigures the reconfigurable partition is referred to as a partial bitstream.

In the scope of this work the static partition will only contain the communication and the reconfiguration components. The actual application of the FPGA will be placed in a single reconfigurable partition. With this setting, it is clear the static partition should be as small possible, thus leaving maximum resources for the actual application. Another consequence of this setting is that every signal to or from the reconfigurable partition has to be routed upon initial configuration.

When the electrical device is turned on, the static partition gets configured with a bitstream which should be encrypted. This is required because the private key of the electrical device is stored in the full bitstream.

B. Bitstream compression

There exist a lot of compression algorithms which offer lossless data compression. It needs no further explication that lossy compression algorithms are unsuitable. For clarity it is noted that the compression ratio (CR) is defined as:

$$CR = \frac{size(compressed\,bitstream)}{size(original\,bitstream)}$$

As proven in [9], additional gains on the CR can be achieved by thoroughly observing what the data represent. For partial

bitstreams this comes down to describing configuration frames, which are the smallest reconfigurable units of an FPGA.

Many algorithms are based on Lempel-Ziv, Huffman encoding and/or run-length encoding. Lempel-Ziv-based algorithms focus on substitutions of repeating occurrences which are maintained in a dictionary [15]. Huffman coding uses short words to represent frequently occurring words, while infrequently occurring words are represented by longer words. For decompression, the Huffman tree needs to be reconstructed. This is considered to be costly [10][13]. Run-length encoding is a very simple technique, which compresses the data by indicating consecutive repetitions in the data. The compressed output will exist of pairs of (w, l) describing the content of the input. The w field defines a word while the l field gives the number of repetitions of this word. All three mentioned algorithms have a fixed word-size in a given compression, but this word-size is adjustable.

C. Communication

To be able to remotely update an FPGA, some form of communication is required. Because this communication channel passes through non-trusted environments, the communication has to be secured. This boils down to offering these concepts:

Entity authentication offers guarantee that the entity with whom one is communicating is in fact the intended entity. Mutual entity authentication is achieved when entity authentication is available in both directions.

Data authentication offers guarantee that the message which is received is in fact the same message as the one that was sent. This concept makes sure any alterations to the message can be detected.

Data confidentiality offers guarantee that the message can only be read by the entities authorized to read it.

In literature lots of cryptographic algorithms are described which offer any combination of the mentioned concepts. In [16], several protocols are thoroughly described. The group of the key establishment protocols result in a secret which is shared between the communicating entities. This secret can be used as a cryptographic key. Although it is an expensive choice with regards to communication and calculation, the Station-To-Station (STS) protocol offers mutual, explicit key authentication and mutual entity authentication when establishing a session key. It is noted that the STS protocol was originally presented more that 25 year ago in [17], but is still considered unbroken and is used all over the world.

D. Security measures

Using the STS protocol instates a number of security measures. With the three concepts mentioned in III-C, vulnerabilities against spoofing attacks, IP theft, man-in-the-middle attacks are covered. The use of freshly generated session keys prevents replay attacks. The fact that the partial bitstream never leaves the FPGA chip is stressed again, because this reduces the fronts on which the scheme can be attack.

E. The single-chip solution

With a static partition that handles the communication protocol and performs all the cryptographic calculations, a partial bitstream can be sent securely to the FPGA. If this bitstream is compressed with a sufficiently large CR, it can be stored on BRAM in the static partition. The static partition, also having access to the reconfiguration primitives, can reconfigure the main application of the FPGA, after decompression and verification.

IV. Implementation

In this section the implementation of a single-chip, secure remote FPGA reconfiguration is described. To the best of our knowledge this is the first implementation of such a scheme. For this implementation it is assumed there is a reconfiguration server which is the only authorized entity to request a reconfiguration. The electrical device, containing the FPGA, is shipped from a trusted environment where the encrypted initial bitstream is stored on the device. Finally, the electrical device is assumed to be connected to the Internet. The implementation described here is built on a ML507 development board by Xilinx, which contains a Xilinx Virtex-5 device (XC5V FX70T). The implementation is made using the Xilinx ISE Design Suite, version 14.5. The choice of the compression algorithm is run-length encoding. The reason for this choice is its simplicity [13], the fact it is not optimized for one FPGA family, and the small decompressor overhead.

A. The static partition

The main task of the static partition is the receiving of a new partial bitstream and the passing down of this bitstream to the Internal Configuration Access Port (ICAP). This task is handled by a MicroBlaze, 32-bit softcore processor. To handle the calculation-intense cryptographic operations several additional hardware components are required, together with an additional internal BRAM component dedicated to temporarily store the incoming bitstream. For this implementation only a single 36 kilobits BRAM is used, which suffices this implementation's need. The remaining BRAMs in the static partition could additionally be used if necessary.

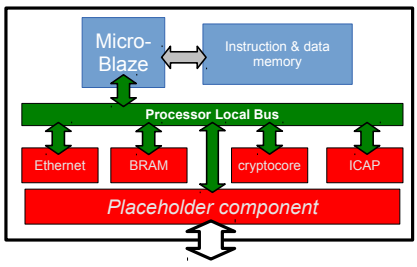

Fig. 1. The interconnected components in the static partition

Fig. 1 shows the MicroBlaze as controlling microprocessor, connected to the IP cores through a processor local bus. The attached cores are: an Ethernet core, the additional BRAM, the cryptocore, an ICAP-core and a placeholder component. With

exception of the cryptocore and the placeholder, all IP cores are default IP cores, available in the development environment.

Because the tool flow for partial reconfiguration implies that the static system is the top-level of the design, a placeholder has to be added. This placeholder is an empty IP component instantiating a black-box entity which will be occupied by the reconfigurable partition. Due to this approach every signal which is required in the reconfigurable partition has be routed to the black-box entity. Although this would request thorough planning on current and future inputs and outputs, it may be avoided by routing **all** signals, except those required for the static partition, to the reconfigurable partition. For the presented implementation only the general purpose IO signals are routed to the reconfigurable partition, together with the clock and reset signals.

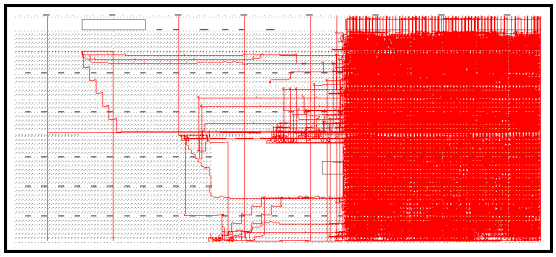

Fig. 2. A rotated floorplan of the FPGA with placed and routed static partition

The placed and routed static partition is visualized in Fig. 2. This image shows that all resources, occupied by the static system, are placed on the south side of the device with only a few lines going into the reconfigurable partition. These lines represent nets to connect certain IO pins located in the reconfigurable partition to the static partition and the nets between the static and reconfigurable partitions.

1) The software: As mentioned above, the MicroBlaze will handle the communication. The UDP/IP protocol is chosen which is commonly supported and has a more lightweight stack then TCP/IP. The software firstly performs an initialization of the IP cores and secondly starts polling the Ethernet core. After handling incoming ARP requests, the software looks for IPV4 packets which are destined for the electrical device.

Packets of which the UDP payload starts with a hexadecimal character "73" belong to the STS-protocol. After finishing this protocol, a session key is established between the FPGA and the configuration server. Packets starting with a hexadecimal "64" belong to the reconfiguration protocol. The partial bitstream is assumed to be larger than the maximum payload of a single UDP frame. Because of this, the partial bitstream is split in chunks. The software on the MicroBlaze accepts three requests: 1) the reception of a new chunk, 2) the verification of the received chunks and 3) the request to reconfigure the FPGA. A more detailed explanation is given in sect. IV-B.

2) The cryptocore: The cryptocore is a custom built IP-core which offers four cryptographic building blocks: an AES-128 component, a SHA-256 component, a random number generator (RNG) and an elliptic curve processor (ECP). All these components are implemented as described in respectively: [18][19][20][21]. Every component has a focus on area, thus contributing to keeping the static partition small. For the sake of completeness it is mentioned that the mode of operation used around the symmetric key cipher is the output feedback mode (OFB) [22]. The four components are interconnected through a single BRAM, and are controlled by a command register and a status register. The block diagram of the complete cryptocore is shown in Fig. 3.

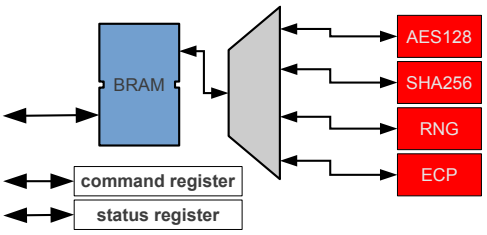

Fig. 3. The block diagram of the cryptocore

For more details on the cryptocore we refer to [8] and [21].

B. The reconfiguration protocol

As already mentioned in sect. IV-A1, the partial bitstream is assumed to consist of multiple chunks. For Ethernet, not using Gigabit Ethernet, the payload of a single packet is 1500 bytes. Because of encapsulation 20 bytes are lost to the IP header and 8 bytes are lost to the UDP header. The reconfiguration protocol used in this work costs another 9 bytes. The first byte is to indicate whether a UDP-frame is part of the STS-protocol or the reconfiguration protocol. The next byte indicates in which phase of the protocol this frame is to be positioned. The following two bytes contain an order number, because with the UDP protocol, out-of-order receiving of frames is possible. The order number gives the relative position of the chunk in the entire bitstream. The next byte provides the word-size while the last two bytes provide the length-size, used in the run-length encoding. Finally, with all encapsulation in place the maximum payload is 1465 bytes.

To be able to validate the integrity of a single chunk a message authentication code (MAC) is added to every chunk. The algorithm used to calculate the MAC is HMAC [23], because an implementation of a hash function is already available in the cryptocore. Because a single chunk will be encrypted, together with a MAC of h bytes, using AES with a block-size of b bytes, the maximal, optimized chunk size c is $payload - h - [(payload - h) mod\ b]$. With a payload of 1465 bytes, a 256-bit MAC and a block-size of 16 bytes for AES, the chunk size is 1424 bytes.

C. Requesting an update

To start an update, the STS-protocol has to be completed, first. This results in a 256-bit session key which gets divided into a key for encryption and a key for message authentication. Subsequently the reconfiguration server has to prepare the bitstream for dispatch. This is visualized in Fig. 4.

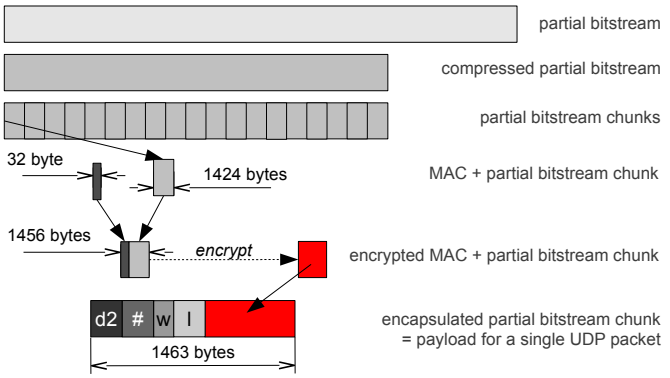

Fig. 4. Partial bitstream segmentation for communication over Ethernet

In a first step the partial bitstream is compressed with the run-length encoding compression algorithm. Because this is a simple algorithm, an exhaustive exploration can be done on the reconfiguration server to find the word and length sizes resulting in the best compression for that specific partial bitstream. Secondly the compressed partial bitstream is divided in chunks of 1424 bytes. These chunks are preceded with a 32-byte MAC on the chunk. Next these 1456-byte chunks are encrypted using AES-128 in OFB mode which results in an encrypted chunk of 1456 bytes. The complete cipher-text is finally preceded with the 7 bytes reconfiguration header of which the second byte is "$0x32$" to request the reception of a new chunk. This results in a 1463 bytes message, which gets encapsulated with the UDP and IPV4 headers to finally result in a 1491 payload for the Ethernet frame. The third step is to send every chunk to the electrical device. The FPGA stores every chunk in the dedicated BRAM, on a position determined by the 2-byte offset in the reconfiguration header. When all chunks are sent, the reconfiguration server should ask for a verification in a fourth step. To do so, a MAC on the entire compressed bitstream is to be calculated and preceded with "$0x6433$". After receiving such a request, the FPGA: 1) decrypts every chunk, 2) verifies every MAC on every chunk and 3) verifies the MAC on the entire bitstream. If all verifications succeed, an internal flag on the MicroBlaze gets set and the success is reported back to the reconfiguration server. Subsequently the dedicated BRAM gets reorganized to have a continuous plaint-text bitstream in memory, by removing all MACs. Fig. 5 shows the evolution of the dedicated BRAM throughout a reconfiguration request. In the fifth and final step, a request is sent to reconfigure the FPGA. This request only gets granted when the internal flag is set, which ensures that a complete and validated bitstream is present on the FPGA, which prevents hitches or incomplete reconfigurations. Moreover, because the complete bitstream is already at the FPGA, the downtime of the main application is kept at a minimum.

D. Results

The implementation as described above is completely implemented and tested. The resource usage of this implementation is presented in Table I.

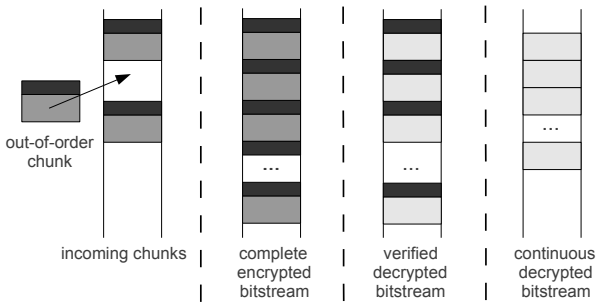

Fig. 5. Evolution of the dedicated BRAM throughout a reconfiguration

component	#Slices	#BRAM	#DSP48E
Used by ECP	866	10	2
Used by SHA256	602	0	0
Used by RNG	294	0	0
Used by AES128	979	4	0
Used by cryptocore	2985	15	2
Used by static partition	4017	50	5
Available in static partition	4560	60	40
Available in reconfigurable partition	6640	88	80
Available in XC5V FX70T	11200	148	128

TABLE I
RESOURCE USAGE OF THE CRYPTOGRAPHIC COMPONENTS, THE CRYPTOCORE AND BOTH PARTITIONS

With a relative usage of around 35%, Table I shows the cost of the static partition is high. The cryptocore is designed with a focus on area and is relatively small [8]. The additional resources used by the static partition are spent on the MicroBlaze with the accompanying bus and the remaining IP cores. Building a lightweight system-on-chip for the static partition could free more resources for the reconfigurable partition.

In sect. IV-A a single BRAM was provided to store the partial bitstream. Table I indicates this can be increased with ten additional BRAMs, resulting in the ability to store a compressed bitstream up to 396 kilobits, which may suffice for small applications with no initialized BRAM. With the segmentation as described in this implementation, uncompressed partial bitstreams are 1588 kilobytes. This implies that the CR should achieve 25% which is quite high, but is achievable if the resource occupation is lenient. If certain applications require more storage, the boundaries between the static and the reconfigurable partition should be shifted.

Table I also shows that the main application still has an acceptable amount of resources to occupy on this FPGA.

E. Improvements on the solution

Two parameters that could improve the presented solution are the size of the MAC and an inter-chunk adjustable word and length size. If the size of the MAC could shrink, the dedicated memory to store a partial bitstream could be used more efficiently. Although not yet implemented, the chunk header offers the possibility to adjust the word and length size between chunks. Additional to the gains in CR, the drawbacks from compressing high-entropy data will have a smaller impact because performing run-length encoding on high-entropy data

will increase the data size rather than compress it.

V. Conclusion and future work

1) Security: The communication channel is the connection between the reconfiguration server and the electrical device. It is assumed this channel is observable by anyone and the message is adaptable by anyone. The STS protocol ensures data authentication, data confidentiality and mutual entity authentication. The chunks are decrypted on the FPGA chip and never leave this chip, in contrast with other work that targets general FPGA reconfiguration. When the MAC on the entire bitstream is validated, it can be assumed that every bitstream chunk has arrived at the FPGA and that the initial order of the chunks has been restored correctly. The MAC on a single chunk can be used to flag to the reconfiguration server which chunk is failing, but this is not implemented yet. Because the FPGA actively participates in the establishment of the session key and contributes to the key material, replay attacks are covered.

2) Single-chip solution: This work combines a simple data compression technique and remote reconfiguration techniques to obtain a single-chip solution for the remote reconfiguration of FPGAs. With the static partition responsible for the secure communication and reconfiguration of the partial bitstream, the single reconfigurable partition holds the main application of the FPGA. Efforts have been made to keep the static partition as small as possible thus leaving the reconfigurable partition to a maximum size. An implementation of this solution is built and tested, and proves the feasibility of this single chip solution, which is, to the best of our knowledge, the first single chip solution for general FPGA reconfiguration. The solution also prevents the loading of incomplete bitstreams and reduces the downtime of the main application.

3) Future work: The authors are planning to work on three fronts. First is the assimilating of state-of-the-art compression techniques. Secondly, improvements on shrinking the static partition are to be explored, and finally an optimum towards hardware-software co-design is to be looked for. The latter is to be done by moving cryptographic algorithms from hardware to software while keeping the additional BRAM-cost acceptable.

Acknowledgement

This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007), by the Flemish Government, FWO G.0550.12N and G.00130.13N, by the Hercules Foundation AKUL/11/19, by the European Commission through the ICT programme under FP7-ICT-2011-8, and by the IWT-TETRA project: "6LoWPAN: Towards zero-configuration for wireless building automation" (120105).

References

[1] M. Hübner, M. Ullmann, F. Weissel, and J. Becker, "Real-time Configuration Code Decompression for Dynamic FPGA Self-Reconfiguration," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS).*, 2004, pp. 138–143.

[2] J. Castillo, P. Huerta, V. Lopez, and J. I. Martinez, "A secure self-reconfiguring architecture based on open-source hardware," in *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (RECONFIG)*. IEEE, 2005, pp. 7–10.

[3] I. Gonzalez, S. Lopez-Buedo, and F. J. Gomez-Arribas, "Implementation of secure applications in self-reconfigurable systems," *Microprocessors and Microsystems*, vol. 32, no. 1, pp. 23 – 32, 2008.

[4] F. Devic, L. Torres, and B. Badrignans, "Secure Protocol Implementation for Remote Bitstream Update Preventing Replay Attacks on FPGA," in *Proceedings of the 2010 International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2010, pp. 179–182.

[5] F. Devic, L. Torres, J. Crenne, B. Badriganans, and P. Benoît, "Secure DPR: Secure update preventing replay attacks for dynamic partial reconfiguration," in *Proceedings of 22nd International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2012, pp. 57–62.

[6] K. Kepa, F. Morgan, K. Kosciuszkiewicz, and T. Surmacz, "SeReCon: a secure reconfiguration controller for self-reconfigurable systems," *International Journal of Critical Computer-Based Systems*, vol. 1, no. 1/2/3, pp. 86–103, 2010.

[7] S. Drimer and M. G. Kuhn, "A Protocol for Secure Remote Updates of FPGA Configurations," in *Proceedings of the 5th International Workshop on Reconfigurable Computing: Architectures, Tools and Applications (ARC)*, ser. Lecture Notes in Computer Science, vol. 5453. Springer, 2009, pp. 50–61.

[8] A. Braeken, J. Genoe, S. Kubera, N. Mentens, A. Touhafi, I. Verbauwhede, Y. Verbelen, J. Vliegen, and K. Wouters, "Secure remote reconfiguration of an FPGA-based embedded system," in *Proceedings of the 6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoc)*. IEEE, 2011, pp. 74–79.

[9] J. Hwa Pan, T. Mitra, and W.-F. Wong, "Configuration Bitstream Compression for Dynamically Reconfigruable FPGAs," in *Proceedings of the International Conference on Computer Aided Design (ICCAD)*. IEEE Computer Society / ACM, 2004, pp. 766 – 773.

[10] D. Koch, C. Beckhoff, and J. Teich, "Bitstream Decompression for High Speed FPGA Configuration from Slow Memories," in *International Conference on Field-Programmable Technology (ICFPT)*, 2007, p. 8.

[11] G. Haiyun and C. Shurong, "Partial Reconfiguration Bitstream Compression for Virtex FPGAs," in *Congress on Image and Signal Processing (CISP)*, vol. 5. IEEE Computer Society, 2008, pp. 183–185.

[12] A. Dandalis and V. K. Prasanna, "Configuration compression for FPGA-based embedded systems," in *Proceedings of the 9th International symposium on Field programmable gate arrays*. ACM, 2001, p. 10.

[13] R. Ştefan and S. D. Coţofană, "Bitstream compression techniques for Virtex 4 FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2008, pp. 323–328.

[14] X. Qin, C. Muthry, and P. Mishra, "Decoding-Aware Compression of FPGA Bitstreams," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 3, pp. 411–419, 2011.

[15] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[16] A. J. Menezes, P. C. v. Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, 5th ed. CRC Press, 1996.

[17] B. O'Higgins, W. Diffie, L. Strawczynski, and R. do Hoog, "Encryption and ISDN - A Natural Fit," in *Proceedings of the International Switching Symposium (ISS)*, vol. 3, 1987.

[18] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer-Verlag, 2002.

[19] D. Rykx and J. Thielen, "Evaluatie van nieuwe hashfunctie kandidaten op FPGA," Master's thesis, KHLim, 2012, S. Indesteege and N. Mentens (promotors).

[20] K. Wold and C. H. Tan, "Analysis and Enhancement of Random Number Generator in FPGA Based on Oscillator Rings," *International Journal of Reconfigurable Computing*, vol. 2009, no. 501672, pp. 385–390, 2009.

[21] J. Vliegen, N. Mentens, J. Genoe, A. Braeken, S. Kubera, A. Touhafi, and I. Verbauwhede, "A compact FPGA-based architecture for elliptic curve cryptography over prime fields," in *Proceedings of the 21st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2010, pp. 313–316.

[22] NIST, "Recommendation for Block Cipher Modes of Operation (NIST SP - 800-38A)," 2001.

[23] NIST, "The Keyed-Hash Message Authentication Code (HMAC) (NIST FIPS - 198-1)," 2008.